UNITED STATES PATENT AND TRADEMARK OFFICE

MN

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/643,586	08/18/2003	Gregory J. Faanes	1376.699US1	4007

21186 7590 05/02/2007
SCHWEGMAN, LUNDBERG, WOESSNER & KLUTH, P.A.
P.O. BOX 2938
MINNEAPOLIS, MN 55402

EXAMINER	
FENNEMA, ROBERT E	
ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
05/02/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	**Application No.** 10/643,586	**Applicant(s)** FAANES ET AL.
	Examiner Robert E. Fennema	**Art Unit** 2183

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 16 February 2007.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-20 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-20 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-20 have been considered. Claims 17-20 have been added as per Applicant's request. Claims 1, 3, 5, and 7-11 have been amended as per Applicant's request.

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

3. Claims 1-4, 7, and 12-18 are rejected under 35 U.S.C. 103(a) as being unpatentable over Beard et al. (USPN 5,430,884, herein Beard), in view of Patterson et al. (herein Patterson).

4. As per Claim 1, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit (Column 3, Lines 1-7), wherein the vector processing unit includes a vector dispatch unit (Column 3 Lines 27-40), decoupling operation of the scalar processing unit from that of the vector processing unit, the method comprising:

dispatching a vector instruction from the scalar processing unit to the vector dispatch unit, (Column 3, Lines 27-30);

reading scalar operands from the scalar processing unit, wherein reading includes transferring the scalar operands from the scalar processing unit to the vector dispatch unit (Column 3, Lines 38-40 and Column 12, Lines 25-36);

predispatching, within the vector dispatch unit, the vector instruction received from the scalar processing unit if all previously received vector instructions are scalar committed; (Column 3, Lines 27-30);

dispatching the predispatched vector instruction from the vector dispatch unit_if all required scalar operands are ready (Column 3, Lines 30-34); and

executing the vector instruction dispatched from the vector dispatch unit as a function of the scalar operands (Column 12, Lines 26-40 disclose the scalar operands being put into a queue from the S registers. Generally in computing systems, the result of a scalar operation is not available outside the pipeline until it is committed, and there is no evidence in the specification to say otherwise, therefore the scalar operand in the queue would not be there until it is "scalar committed", which then allows the vector instruction to initiate, or begin execution), but fails to teach:

wherein dispatching includes sending the vector instruction from the scalar processing unit to the vector dispatch unit even if all scalar operands are not ready.

Beard teaches the decoupling operation as claimed above, but does not teach that the vector instruction can be sent to the vector dispatch unit even if all scalar operands are not ready (and in fact Beard teaches that it is sent to the vector dispatch unit when all scalar operands are accounted for). However, Patterson teaches a reservation station, which has a purpose of holding instructions prior to execution when the operands have not yet been made available (Page 252). The advantage of a reservation station is to eliminate WAW (write after write) and WAR (write after read) hazards, to provide scoreboarding so operands can be sent to the instruction before commitment, and also to allow other instructions to be fetched instead of holding it in the scalar stage. Given these advantages, one of ordinary skill in the art would have been motivated to make use of a reservation station in Beard's invention, and the reservation station is analogous to vector initiation queue, with the enhancement of being able to hold instructions which do not yet have values for their operands. Therefore, given the advantages of eliminating hazards and potentially reducing performance by holding vector instructions in the scalar processing unit, one of ordinary skill in the art at the time the invention was made would have been motivated to make use of a reservation station to hold pending instructions instead of the vector initiation queue as taught by Beard, allowing vector instructions to pass directly into it, while still being able to perform the necessary steps disclosed by Beard in order to execute the invention.

5. As per Claim 2, Beard teaches: The method according to claim 1, wherein executing the dispatched vector instruction includes translating an address associated with the vector instruction (Column 31, Lines 3-8), but fails to teach:

and trapping on a translation fault.

While Beard does not explicitly teach trapping on a translation fault, Patterson teaches that a way to save the pipeline state safely in the event of an exception (also known as a fault) is to insert a trap in the pipeline (Page 183). A translation fault is an example of an exception, and thus would require a trap in order for the processor to regain correct operating status. One of ordinary skill in the art would recognize the use of a trap in order to regain control of the system, which is required in order for the computer system to continue operating properly. Therefore, one of ordinary skill in the art at the time the invention was made would have recognized the need to trap in the event of a translation fault.

6. As per Claim 3, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, wherein the vector processing unit includes a vector dispatch unit,, a method of decoupling operation of the scalar processing unit from that of the vector processing unit, the method, comprising:

dispatching a vector instruction from the scalar processing unit to the vector dispatch unit (Column 3, Lines 27-30);

reading scalar operands from the scalar processing unit, wherein reading includes transferring the scalar operands from the scalar processing unit to the vector dispatch unit (Column 11, Lines 44-53);

predispatching, within the vector dispatch unit, the vector instruction received from the scalar processing unit if all previously received vector instructions are scalar committed_(Column 3, Lines 27-30);

dispatching the predispatched vector instruction from the vector dispatched unit_if all required scalar operands are ready (Column 3, Lines 30-34);

generating an address for a vector load (Column 12, Lines 40-44 show how an address can be generated);

issuing a vector load request to memory (Column 23, Lines 24-26);

receiving vector data from memory (Column 23, Lines 24-26, when a load is requested from memory, it has to be received);

executing the vector instruction <u>dispatched from the vector dispatch unit</u> on the vector data stored in the vector register (Column 3, Lines 27-40), but fails to teach:

wherein dispatching includes sending the vector instruction from the scalar processing unit to the vector dispatch unit even if all scalar operands are not ready;

storing the vector data in a load buffer;

transferring the vector data from the load buffer to a vector register.

Beard teaches the decoupling operation as claimed above, but does not teach that the vector instruction can be sent to the vector dispatch unit even if all scalar operands are not ready (and in fact Beard teaches that it is sent to the vector dispatch unit when all scalar operands are accounted for). However, Patterson teaches a reservation station, which has a purpose of holding instructions prior to execution when the operands have not yet been made available (Page 252). The advantage of a reservation station is to eliminate WAW (write after write) and WAR (write after read) hazards, to provide scoreboarding so operands can be sent to the instruction before commitment, and also to allow other instructions to be fetched instead of holding it in the scalar stage. Given these advantages, one of ordinary skill in the art would have been motivated to make use of a reservation station in Beard's invention, and the reservation station is analogous to vector initiation queue, with the enhancement of being able to hold instructions which do not yet have values for their operands. Therefore, given the advantages of eliminating hazards and potentially reducing performance by holding vector instructions in the scalar processing unit, one of ordinary skill in the art at the time the invention was made would have been motivated to make use of a reservation station to hold pending instructions instead of the vector initiation queue as taught by Beard, allowing vector instructions to pass directly into it, while still being able to perform the necessary steps disclosed by Beard in order to execute the invention.

While Beard teaches an instruction cache, which consists of multiple buffers (Column 7, Lines 52-55), he does not explicitly teach a data cache, which is commonly used in computer systems to vastly increase performance in loads and stores. Patterson teaches that most computers have 2 levels of caches now, both instruction and data caches, with Pages 380-381 showing an example data cache. As it is well known in the art, the cache would read the data directly from memory (or from a higher cache level), and the computer system would then read the data from the cache (buffer) into the appropriate register(s). One of ordinary skill in the art would have recognized the need to add a data cache into Beards system, for the same reason there is an instruction cache, increased performance and faster memory reads. Therefore, one of ordinary skill in the art at the time the invention was made would have added a data cache (which can function as a load buffer) into Beards invention to increase performance.

7. As per Claim 4, Beard teaches: The method according to claim 3, wherein the vector processing unit includes a vector execute unit (Column 3, Lines 1-7, both the scalar and vector processor has functional units, which execute instructions) and a vector load/store unit (Patterson, Page B-5, the vector load-store unit), wherein issuing a vector load request to memory includes issuing and executing vector memory references in the vector load/store unit when the vector load store unit has received the instruction and memory operands from the scalar processing unit (Column 3, Lines 27-40). While Beard did not explicitly disclose a vector load/store unit, Patterson discloses that it is an essential unit of a basic vector architecture to load and store to/from vectors. Therefore, a vector load/store unit would necessarily have to be present in Beards invention in order for it to function.

8. As per Claim 7, Beard teaches: A computer system, comprising:

a scalar processing unit (Column 3, Lines 1-7); and

a vector processing unit (Column 3, Lines 1-7) wherein the vector processing unit includes a vector dispatch unit, a vector execute unit and a vector load/store unit;

wherein the scalar processing unit dispatches a vector instruction to the vector dispatch unit, (Column 3, Lines 27-30);

wherein the scalar processing unit reads scalar operands and transfers the read scalar operands from the scalar processing unit to the vector dispatch unit (Column 3, Lines 21-30 and 38-40);

wherein the vector dispatch unit predispatches, wherein the vector dispatch unit, the vector instruction received from the scalar processing unit if all previously received vector instructions are scalar committed (Column 3, Lines 27-30) and then dispatches the predispatched vector instruction from the vector dispatch unit to one or more of the vector execute unit and the vector load/store unit if all required scalar operands are ready (Column 3, Lines 30-34);

wherein the vector load/store unit receives an instruction and memory operands from the scalar processing unit, issues and executes a vector memory load reference as a function of the instruction and the memory operands receives from the scalar processing unit (Column 3 Lines 27-40), but fails to teach:

dispatching even if all scalar operands are not ready;

and stores data received as a result of the vector memory reference in a load buffer; and

wherein the vector execute unit issues the vector memory load instruction and transfers the data received as a result of the vector memory reference from the load buffer to a vector register.

Beard teaches the decoupling operation as claimed above, but does not teach that the vector instruction can be sent to the vector dispatch unit even if all scalar operands are not ready (and in fact Beard teaches that it is sent to the vector dispatch unit when all scalar operands are accounted for). However, Patterson teaches a reservation station, which has a purpose of holding instructions prior to execution when the operands have not yet been made available (Page 252). The advantage of a reservation station is to eliminate WAW (write after write) and WAR (write after read) hazards, to provide scoreboarding so operands can be sent to the instruction before commitment, and also to allow other instructions to be fetched instead of holding it in the scalar stage. Given these advantages, one of ordinary skill in the art would have been motivated to make use of a reservation station in Beard's invention, and the reservation station is analogous to vector initiation queue, with the enhancement of being able to hold instructions which do not yet have values for their operands. Therefore, given the advantages of eliminating hazards and potentially reducing performance by holding vector instructions in the scalar processing unit, one of ordinary skill in the art at the time the invention was made would have been motivated to make use of a reservation station to hold pending instructions instead of the vector initiation queue as taught by Beard, allowing vector instructions to pass directly into it, while still being able to perform the necessary steps disclosed by Beard in order to execute the invention.

While Beard teaches an instruction cache, which consists of multiple buffers (Column 7, Lines 52-55), he does not explicitly teach a data cache, which is commonly used in computer systems to vastly increase performance in loads and stores. Patterson teaches that most computers have 2 levels of caches now, both instruction and data caches, with Pages 380-381 showing an example data cache. As it is well known in the art, the cache would read the data directly from memory (or from a higher cache level), and the computer system would then read the data from the cache (buffer) into the appropriate register(s). One of ordinary skill in the art would have recognized the need to add a data cache into Beards system, for the same reason there is an instruction cache, increased performance and faster memory reads. Therefore, one of ordinary skill in the art at the time the invention was made would have added a data cache (which can function as a load buffer) into Beards invention to increase performance.

9. As per Claim 12, Patterson teaches: The method according to claim 3, wherein storing the vector data in a load buffer and transferring the vector data from the load buffer to a vector register are decoupled from each other (Pages 375-381, the cache load is decoupled from a cache read).

10. As per Claim 13, Beard teaches: The method according to claim 3, but fails to teach: wherein storing the vector data in a load buffer includes writing memory load data to the load buffer until all previous memory operations complete without fault. However, Patterson teaches that out of order execution adds a large amount of complexity to a system, and that a large amount of hardware is required to handle it (Pages 241-243). Given the disadvantages of out-of-order execution, one of ordinary skill in the art would have been motivated to remove the out of order elements of the system, and make it a simpler in-order execution machine in order to reduce complexity and hardware cost, which would require all previous memory operations to complete before the current one could begin.

11. As per Claim 14, Patterson teaches: The method according to claim 4, wherein storing the vector data in a load buffer and transferring the vector data from the load buffer to a vector register are decoupled from each other (Pages 375-381, the cache load is decoupled from a cache read).

12. As per Claim 15, Patterson teaches: The method according to claim 4, wherein storing the vector data in a load buffer includes writing memory load data to the load buffer until all previous memory operations complete without fault (Pages 241-243 teaches out-of-order execution and its disadvantages, providing a motivation to use in order execution instead, as described on Page 241).

13. As per Claim 16, Patterson teaches: The system according to claim 7, wherein the load buffer stores memory load data until it is determined that no previous memory operation will fail and, if no previous memory operations have failed, the load buffer transfers the data to the vector register (Pages 241-243 teaches out-of-order execution and its disadvantages, providing a motivation to use in order execution instead as described on Page 241).

14. As per Claim 17, Beard teaches: The method according to claim 1, wherein the method further includes: marking the vector instruction as complete, wherein marking the vector instruction as complete includes:

if the vector instruction is not a memory instruction and if the vector instruction does not require scalar operands, indicating that the vector instruction is complete when the vector instruction is dispatched from the scalar processing unit (Column 3, Lines 27-30, the system knows when the scalar operands (if any) are available, thus being "complete");

if the vector instruction is not a memory instruction but requires scalar operands, indicating that the vector instruction is complete when the scalar operands are available (Column 3, Lines 27-30); and

if the vector instruction is a memory instruction, indicating that the vector instruction is complete when the vector address has been translated (Column 5, Lines 51-54); and

graduating the vector instruction if the vector instruction is marked complete (Column 3, Lines 61-67, initiation).

15. As per Claim 18, Beard teaches: The method according to claim 7, wherein the method further includes: marking the vector instruction as complete, wherein marking the vector instruction as complete includes:

if the vector instruction is not a memory instruction and if the vector instruction does not require scalar operands, indicating that the vector instruction is complete when the vector instruction is dispatched from the scalar processing unit (Column 3, Lines 27-30, the system knows when the scalar operands (if any) are available, thus being "complete");

if the vector instruction is not a memory instruction but requires scalar operands, indicating that the vector instruction is complete when the scalar operands are available (Column 3, Lines 27-30); and

if the vector instruction is a memory instruction, indicating that the vector instruction is complete when the vector address has been translated (Column 5, Lines 51-54); and

graduating the vector instruction if the vector instruction is marked complete (Column 3, Lines 61-67, initiation).

16. Claims 5-6, 8-11 and 19-20 are rejected under 35 U.S.C. 103(a) as being unpatentable over Beard and Patterson, further in view of Gharachorloo et al (herein Gharachorloo).

17. As per Claim 5, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, wherein the vector processing unit includes a vector dispatch unit, a method of decoupling operation of the scalar processing unit from that of the vector processing unit, the method comprising:

dispatching a vector instruction from the scalar processing unit to the vector dispatch unit, (Column 3, Lines 27-30);

reading scalar operands from the scalar processing unit, wherein reading includes transferring the scalar operands from the scalar processing unit to the vector dispatch unit (Column 11, Lines 44-53);

predispatching, within the vector dispatch unit, the vector instruction received from the scalar processing unit if all previously received vector instructions are scalar committed (Column 3, Lines 27-30);

dispatching the predispatched vector instruction from the vector dispatched unit if all required scalar operands are ready (Column 3, Lines 30-34);

generating a first and a second address for a vector load (Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit, and Column 3 Lines 1-3 is one of many examples showing Beards invention is capable of executing multiple instructions, and Column 12, Lines 26-28 show that a load can have multiple addresses);

issuing first and second vector load requests to memory (Column 23, Lines 24-26);

receiving vector data associated with the first and second addresses from memory (Column 23, Lines 24-26, when a load is requested from memory, it has to be received);

storing vector data associated with the first address in a first vector register (Column 23, Lines 24-33);

storing vector data associated with the second address in a second vector register (Column 23, Lines 24-33);

executing a vector instruction on the vector data stored in the first vector register (Column 3, Lines 27-40);

executing the vector instruction dispatched from the vector dispatch unit on the vector data stored in the second vector register (Column 3, Lines 27-40), but fails to teach:

wherein dispatching includes sending the vector instruction from the scalar processing unit to the vector dispatch unit even if all scalar operands are not ready;

renaming the second vector register.

Beard teaches the decoupling operation as claimed above, but does not teach that the vector instruction can be sent to the vector dispatch unit even if all scalar operands are not ready (and in fact Beard teaches that it is sent to the vector dispatch unit when all scalar operands are accounted for). However, Patterson teaches a reservation station, which has a purpose of holding instructions prior to execution when the operands have not yet been made available (Page 252). The advantage of a reservation station is to eliminate WAW (write after write) and WAR (write after read) hazards, to provide scoreboarding so operands can be sent to the instruction before commitment, and also to allow other instructions to be fetched instead of holding it in the scalar stage. Given these advantages, one of ordinary skill in the art would have been motivated to make use of a reservation station in Beard's invention, and the reservation station is analogous to vector initiation queue, with the enhancement of being able to hold instructions which do not yet have values for their operands. Therefore, given the advantages of eliminating hazards and potentially reducing performance by holding vector instructions in the scalar processing unit, one of ordinary skill in the art at the time the invention was made would have been motivated to make use of a reservation station to hold pending instructions instead of the vector initiation queue as taught by Beard, allowing vector instructions to pass directly into it, while still being able to perform the necessary steps disclosed by Beard in order to execute the invention.

While Beard does not explicitly teach renaming, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. The reorder buffer that comes with that eliminates storage conflicts through register renaming (Page 5, Section 4.2). The reorder buffer also allows for storage of speculative results, to potentially increase the workload of the processor. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer (and thus being able to rename registers) to implement the out-of-order execution in Beards invention to increase performance.

18. As per Claim 6, Beard teaches: The method according to claim 5, wherein the vector processing unit includes a vector execute unit (Column 3, Lines 1-7, both the scalar and vector processor has functional units, which execute instructions) and a vector load/store unit (Patterson, Page B-5, the vector load-store unit), wherein issuing a vector load request to memory includes issuing and executing vector memory references in the vector load/store unit when the vector load store unit has received the instruction and memory operands from the scalar processing unit (Column 3, Lines 27-40). While Beard did not explicitly disclose a vector load/store unit, Patterson discloses that it is an essential unit of a basic vector architecture to load and store to/from vectors. Therefore, a vector load/store unit would necessarily have to be present in Beards invention in order for it to function.

19. As per Claim 8, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, a method of decoupling scalar and vector execution, comprising:

dispatching a vector instruction that requires scalar operands from the scalar processing unit to the scalar instruction queue and to a vector instruction queue (The scalar required part of the instruction goes into the scalar queue (Column 3, Lines 27-30);

executing the vector instruction in the scalar processing unit, wherein executing the vector instruction in the scalar processing unit includes generating an address and writing the address to a scalar operand queue (Column 3, Lines 38-40. In addition, Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit);

predispatching, within the vector dispatch unit, the vector instruction received from the scalar processing unit if all previously received vector instructions are scalar committed (Column 3, Lines 27-30);

notifying the vector processing unit that the address is available in the scalar operand queue (Column 11, Lines 44-53. It is notified by a bit in the scalar register scoreboard. When the vector processing unit looks into the scoreboard, it can determine if the scalar operand is available);

dispatching the predispatched vector instruction from the vector dispatch unit if all required scalar operands are ready (Column 3, Lines 30-34); and

executing the vector instruction dispatched from the vector dispatch unit in the vector processing unit, wherein executing the vector instruction in the vector processing unit includes reading the address from the scalar operand queue and generating a memory request as a function of the address read from the scalar operand queue (Column 12, Lines 26-36, the scalar operand data is transferred to the vector register unit as it begins execution. In addition, Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit), but fails to teach:

wherein dispatching includes sending the vector instruction from scalar processing unit to the vector dispatch unit even if all scalar operands are not ready;

dispatching scalar instructions to a scalar instruction queue in the scalar processing unit.

Beard teaches the decoupling operation as claimed above, but does not teach that the vector instruction can be sent to the vector dispatch unit even if all scalar operands are not ready (and in fact Beard teaches that it is sent to the vector dispatch unit when all scalar operands are accounted for). However, Patterson teaches a reservation station, which has a purpose of holding instructions prior to execution when the operands have not yet been made available (Page 252). The advantage of a reservation station is to eliminate WAW (write after write) and WAR (write after read) hazards, to provide scoreboarding so operands can be sent to the instruction before commitment, and also to allow other instructions to be fetched instead of holding it in the scalar stage. Given these advantages, one of ordinary skill in the art would have been motivated to make use of a reservation station in Beard's invention, and the reservation station is analogous to vector initiation queue, with the enhancement of being able to hold instructions which do not yet have values for their operands. Therefore, given the advantages of eliminating hazards and potentially reducing performance by holding vector instructions in the scalar processing unit, one of ordinary skill in the art at the time the invention was made would have been motivated to make use of a reservation station to hold pending instructions instead of the vector initiation queue as taught by Beard, allowing vector instructions to pass directly into it, while still being able to perform the necessary steps disclosed by Beard in order to execute the invention.

While Beard does not explicitly teach a queue to put scalar instructions in before execution, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer to implement the out-of-order execution in Beards invention to increase performance.

20. As per Claim 9, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, a method of decoupling scalar and vector execution, comprising:

dispatching a vector instruction from the scalar processing unit that requires scalar operands to the scalar instruction queue and to a vector instruction queue (The scalar required part of the instruction goes into the scalar queue in the vector processing unit (Column 3, Lines 27-30);

executing the vector instruction in the scalar processing unit, wherein executing the vector instruction in the scalar processing unit includes writing a scalar operand to a scalar operand queue (Column 3, Lines 38-40);

predispatching, within the vector processing unit, the vector instruction received from the scalar processing unit if all previously received vector instructions are scalar committed (Column 3, Lines 27-30);

notifying the vector processing unit that the scalar operand is available in the scalar operand queue (Column 11, Lines 44-53. It is notified by a bit in the scalar register scoreboard. When the vector processing unit looks into the scoreboard, it can determine if the scalar operand is available);

dispatching the predispatched vector instruction from the vector processing unit if all required scalar operands are ready (Column 3, Lines 30-34); and

executing the vector instruction *dispatched from the vector processing unit*in the vector processing unit, wherein executing the vector instruction in the vector processing unit includes reading the scalar operand from the scalar operand queue (Column 12, Lines 26-36, the scalar operand data is transferred to the vector register unit as it begins execution), but fails to teach:

wherein dispatching includes sending the vector instruction from the scalar processing unit to the vector instruction queue in the vector processing unit even if all scalar operands are not ready;

dispatching scalar instructions to a scalar instruction queue in the scalar processing unit.

Beard teaches the decoupling operation as claimed above, but does not teach that the vector instruction can be sent to the vector dispatch unit even if all scalar operands are not ready (and in fact Beard teaches that it is sent to the vector dispatch unit when all scalar operands are accounted for). However, Patterson teaches a reservation station, which has a purpose of holding instructions prior to execution when the operands have not yet been made available (Page 252). The advantage of a reservation station is to eliminate WAW (write after write) and WAR (write after read) hazards, to provide scoreboarding so operands can be sent to the instruction before commitment, and also to allow other instructions to be fetched instead of holding it in the scalar stage. Given these advantages, one of ordinary skill in the art would have been motivated to make use of a reservation station in Beard's invention, and the reservation station is analogous to vector initiation queue, with the enhancement of being able to hold instructions which do not yet have values for their operands. Therefore, given the advantages of eliminating hazards and potentially reducing performance by holding vector instructions in the scalar processing unit, one of ordinary skill in the art at the time the invention was made would have been motivated to make use of a reservation station to hold pending instructions instead of the vector initiation queue as taught by Beard, allowing vector instructions to pass directly into it, while still being able to perform the necessary steps disclosed by Beard in order to execute the invention.

While Beard does not explicitly teach a queue to put scalar instructions in before execution, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer to implement the out-of-order execution in Beards invention to increase performance.

21. As per Claim 10, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, a method of executing a vector instruction, comprising:

dispatching a vector instruction from the scalar processing unit to the scalar instruction queue and to vector instruction queue in the vector processing unit (The scalar required part of the instruction goes into the scalar queue, and the vector queue is disclosed in Column 3, Lines 34-36);

executing the vector instruction in the scalar processing unit, wherein executing the vector instruction in the scalar processing unit includes generating an address and writing the address to a scalar operand queue (Column 3, Lines 38-40. In addition, Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit);

predispatching, wherein the vector processing unit, the vector instruction received from the scalar processing unit if all previously received vector instructions are scalar committed (Column 3, Lines 27-30);

notifying the vector processing unit that the address is available in the scalar operand queue (Column 11, Lines 44-53. It is notified by a bit in the scalar register scoreboard. When the vector processing unit looks into the scoreboard, it can determine if the scalar operand is available);

dispatching the predispatched vector instruction from the vector processing unit if all required scalar operands are ready (Column 3, Lines 30-34); and

executing the vector instruction dispatched from the vector processing unit in the vector processing unit (Column 12, Lines 26-36, the scalar operand data is transferred to the vector register unit as it begins execution), wherein executing the dispatched vector instruction in the vector processing unit includes:

reading the address from the scalar operand queue (Column 12, Lines 33-36 show the vector register getting the operand (in this case the address) as it begins);

generating a memory request as a function of the address read from the scalar operand queue (Column 3 Lines 27-40, wherein all memory requests are a function of the address, and all addresses must necessarily come from the scalar unit, and thus the scalar operand queue for the vector to use it. Also see Column 23, Lines 24-26);

receiving vector data from memory (Column 23, Lines 24-26, when a load is requested from memory, it has to be received);

executing a vector instruction on the vector data storing in the vector register (Column 3, Lines 27-40), but fails to teach:

wherein dispatching includes sending the vector instruction from the scalar processing unit to the vector instruction queue in the vector processing unit even if all scalar operands are not ready;

dispatching scalar instructions to a scalar instruction queue in the scalar processing unit;

storing the vector data in a load buffer;

transferring the vector data from the load buffer to a vector register.

Beard teaches the decoupling operation as claimed above, but does not teach that the vector instruction can be sent to the vector dispatch unit even if all scalar operands are not ready (and in fact Beard teaches that it is sent to the vector dispatch unit when all scalar operands are accounted for). However, Patterson teaches a reservation station, which has a purpose of holding instructions prior to execution when the operands have not yet been made available (Page 252). The advantage of a reservation station is to eliminate WAW (write after write) and WAR (write after read) hazards, to provide scoreboarding so operands can be sent to the instruction before commitment, and also to allow other instructions to be fetched instead of holding it in the scalar stage. Given these advantages, one of ordinary skill in the art would have been motivated to make use of a reservation station in Beard's invention, and the reservation station is analogous to vector initiation queue, with the enhancement of being able to hold instructions which do not yet have values for their operands.

Therefore, given the advantages of eliminating hazards and potentially reducing performance by holding vector instructions in the scalar processing unit, one of ordinary skill in the art at the time the invention was made would have been motivated to make use of a reservation station to hold pending instructions instead of the vector initiation queue as taught by Beard, allowing vector instructions to pass directly into it, while still being able to perform the necessary steps disclosed by Beard in order to execute the invention.

While Beard does not explicitly teach a queue to put scalar instructions in before execution, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer to implement the out-of-order execution in Beards invention to increase performance.

Furthermore, while Beard teaches an instruction cache, which consists of multiple buffers (Column 7, Lines 52-55), he does not explicitly teach a data cache, which is commonly used in computer systems to vastly increase performance in loads and stores. Patterson teaches that most computers have 2 levels of caches now, both instruction and data caches, with Pages 380-381 showing an example data cache. As it is well known in the art, the cache would read the data directly from memory (or from a higher cache level), and the computer system would then read the data from the cache (buffer) into the appropriate register(s). One of ordinary skill in the art would have recognized the need to add a data cache into Beards system, for the same reason there is an instruction cache, increased performance and faster memory reads. Therefore, one of ordinary skill in the art at the time the invention was made would have added a data cache (which can function as a load buffer) into Beards invention to increase performance.

22. As per Claim 11, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, a method of unrolling a loop, comprising:

preparing a first and a second vector instruction (Column 3 Lines 1-3 is one of many examples showing Beards invention is capable of executing multiple instructions),

dispatching the first and second vector instructions from the scalar processing unit to a scalar instruction queue in the scalar processing unit and to a vector instruction queue in the vector processing unit (The scalar required part of the instruction does into the scalar queue, and the vector queue is disclosed in Column 3, Lines 34-36), executing a portion of each vector instruction in the scalar processing unit (Column 3, Lines 38-40. In addition, Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit),

predispatching, within the vector processing unit, each vector instruction received from the scalar processing unit if all previously received vector instructions are scalar committed (Column 3, Lines 27-30);

notifying the vector processing unit that the scalar operand is available in the scalar operand queue (Column 11, Lines 44-53. It is notified by a bit in the scalar register scoreboard. When the vector processing unit looks into the scoreboard, it can determine if the scalar operand is available);

dispatching the predispatched vector instruction from the vector processing unit if all required operands are ready (Column 3, Lines 30-34); and

executing the first and second vector instructions dispatched from the vector processing unit in the vector processing unit, wherein executing the dispatched vector instructions in the vector processing unit includes reading the scalar operands associated with each instruction from the scalar operand queue (Column 12, Lines 26-36, the scalar operand data is transferred to the vector register unit as it begins execution, and is done only when each instruction needs it), but fails to teach:

wherein dispatching includes sending the first and second vector instructions from the scalar processing unit to the vector processing unit even if all scalar operands are not ready;

wherein each vector instruction execute an iteration through the loop and wherein each vector instruction requires calculation of a scalar loop value;

wherein executing a portion of each vector instruction in the scalar processing unit includes writing a scalar operand representing the scalar loop value calculated for each vector instruction to a scalar operand queue.

Beard teaches the decoupling operation as claimed above, but does not teach that the vector instruction can be sent to the vector dispatch unit even if all scalar operands are not ready (and in fact Beard teaches that it is sent to the vector dispatch unit when all scalar operands are accounted for). However, Patterson teaches a reservation station, which has a purpose of holding instructions prior to execution when the operands have not yet been made available (Page 252). The advantage of a reservation station is to eliminate WAW (write after write) and WAR (write after read) hazards, to provide scoreboarding so operands can be sent to the instruction before commitment, and also to allow other instructions to be fetched instead of holding it in the scalar stage. Given these advantages, one of ordinary skill in the art would have been motivated to make use of a reservation station in Beard's invention, and the reservation station is analogous to vector initiation queue, with the enhancement of being able to hold instructions which do not yet have values for their operands. Therefore, given the advantages of eliminating hazards and potentially reducing performance by holding vector instructions in the scalar processing unit, one of ordinary skill in the art at the time the invention was made would have been motivated to make use of a reservation station to hold pending instructions instead of the vector initiation queue as taught by Beard, allowing vector instructions to pass directly into it, while still being able to perform the necessary steps disclosed by Beard in order to execute the invention.

While Beard does not explicitly teach a queue to put scalar instructions in before execution, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer to implement the out-of-order execution in Beards invention to increase performance.

While Beard does not also explicitly teach a vector instruction, each of which each vector instruction executes an iteration of a loop, Patterson gives an example of how vector instructions can be used for this purpose (Pages B-7 and B-8). In this example, a MULTSV instruction is used for the first part, and an ADDV instruction for the second, although one of ordinary skill in the art could also see how this would be able to be modified to be done in one or multiple vector instructions, depending on the required steps of the loop in question. It can also be seen that in this example, "a" is a scalar operand required for the vector instructions to execute. One of ordinary skill in the art would be able to see the value in using vector instructions to execute this loop, as shown by the vastly decreased code size as shown in the example on B-8. Therefore, one of ordinary skill in the art at the time the invention was made would have recognized the advantage of using vector instructions for the purpose of executing loops in Beards invention, to decrease code size, and therefore increase performance.

23. As per Claim 19, Beard teaches: The method according to claim 8, wherein the method further includes: marking the vector instruction as complete, wherein marking the vector instruction as complete includes:

if the vector instruction is not a memory instruction and if the vector instruction does not require scalar operands, indicating that the vector instruction is complete when the vector instruction is dispatched from the scalar processing unit (Column 3, Lines 27-30, the system knows when the scalar operands (if any) are available, thus being "complete");

if the vector instruction is not a memory instruction but requires scalar operands, indicating that the vector instruction is complete when the scalar operands are available (Column 3, Lines 27-30); and

if the vector instruction is a memory instruction, indicating that the vector instruction is complete when the vector address has been translated (Column 5, Lines 51-54); and graduating the vector instruction if the vector instruction is marked complete (Column 3, Lines 61-67, initiation).

24. As per Claim 20, Beard teaches: The method according to claim 9, wherein the method further includes: marking the vector instruction as complete, wherein marking the vector instruction as complete includes:

if the vector instruction is not a memory instruction and if the vector instruction does not require scalar operands, indicating that the vector instruction is complete when the vector instruction is dispatched from the scalar processing unit (Column 3, Lines 27-30, the system knows when the scalar operands (if any) are available, thus being "complete");

if the vector instruction is not a memory instruction but requires scalar operands, indicating that the vector instruction is complete when the scalar operands are available (Column 3, Lines 27-30); and

if the vector instruction is a memory instruction, indicating that the vector instruction is complete when the vector address has been translated (Column 5, Lines 51-54); and

graduating the vector instruction if the vector instruction is marked complete (Column 3, Lines 61-67, initiation).

Response to Arguments

25. Applicant's arguments with respect to claims 1-20 have been considered but are moot in view of the new ground(s) of rejection.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Robert E. Fennema whose telephone number is (571) 272-2748. The examiner can normally be reached on Monday-Friday, 8:45-6:15.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Robert E Fennema
Examiner
Art Unit 2183

RF

[illegible]

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100